1. Tempo disponibile 120 minuti.
2. Non è possibile consultare appunti, slide, libri, persone, siti web, ecc.
3. Scrivere in modo leggibile, su ogni foglio, nome, cognome e numero di matricola.
4. Le soluzioni agli esercizi che richiedono di progettare un algoritmo devono:
   - spiegare a parole l'algoritmo (se utile, anche con l'aiuto di esempi o disegni),
   - fornire e commentare lo pseudo-codice (indicando il significato delle variabili),
   - calcolare la complessità (con tutti i passaggi matematici necessari),
   - se l'esercizio ammette più soluzioni, a soluzioni computazionalmente più efficienti e/o concettualmente più semplici sono assegnati punteggi maggiori.

1. Calcolare la complessità $T(n)$ del seguente algoritmo `mystery`:

```
algoritmo mystery(n: Int) --> Int
    if (n<=1) return 1
    else
        int tot = 0, x = n/2
        for i = 1..n
            for j = 1..i
                tot = tot+n
            endfor
        endfor
        tot = tot+mystery(x)+mystery(x)+mystery(x)+mystery(x)
    endif
    return tot
```

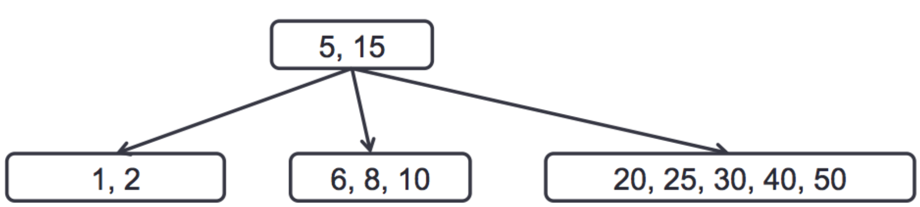
Soluzione Calcoliamo il costo $T(n)$ dell'algoritmo `mystery`. Tale algoritmo è ricorsivo. Nel caso base, $n \leq 1$, ha complessità costante. Nel caso ricorsivo, $n > 1$, effettua quattro chiamate ricorsive con partizioni bilanciate dimezzando il parametro di invocazione, in quanto $x = n/2$, ed esegue due cicli annidati. Il ciclo esterno viene eseguito n volte. Durante la i-esima di queste esecuzioni, il ciclo interno viene eseguito i volte, quindi l'effetto combinato di tali cicli è di eseguire una operazione di costo costante, l'assegnamento della variabile `tot`, una quantità di volte uguale a $\sum_{i=1}^{n} i = \frac{n\times(n+1)}{2} = O(n^2)$. $T(n)$ soddisfa quindi la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} c_1 & \text{se } n \leq 1 \\ 4T(\frac{n}{2}) + c_2 n^2 & \text{altrimenti} \end{cases}$$

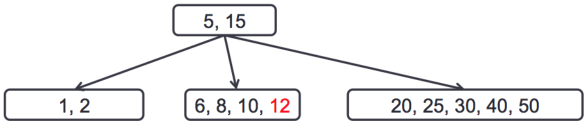
Applicando il Master Theorem, avendo $a = 4$, $b = 2$ (quindi $\alpha = \frac{\log a}{\log b} = 2$) e $\beta = 2$, otteniamo $T(n) = O(n^2 \log n)$ (considerando il secondo caso del teorema).

2. Dato il BTree mostrato sotto di grado `t=3` mostrare lo stato dell'albero dopo ognuna delle seguenti operazioni eseguite in ordine: INS(12), INS(60), CANC(1), CANC(15), CANC(30), INS(1), CANC(60), CANC(25)

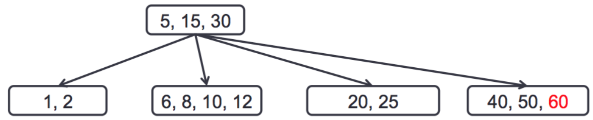
Commentare brevemente le singole operazioni.



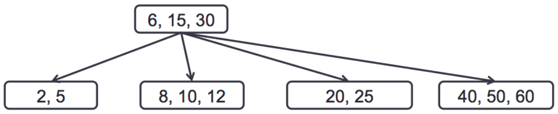
Soluzione Vedi tabella seguente:



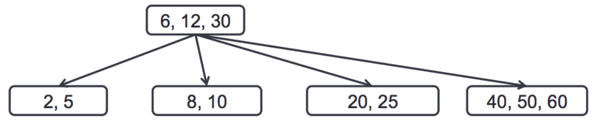
(a) INS(12): Chiave 12 aggiunta al nodo nel rispetto dei vincoli Btree. Non necessarie altre operazioni.



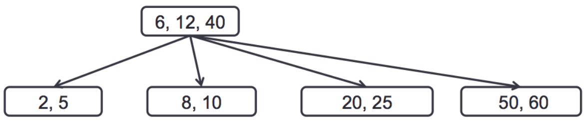
(b) INS(60): Aggiungendo la chiave 60 il vincolo sul numero di chiavi massime in un nodo non è rispettato, necessaria un'operazione di split.



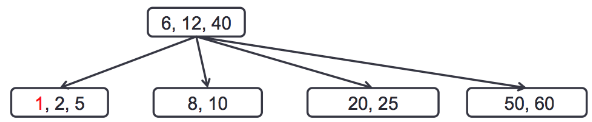
(c) CANC(1): Cancellando la chiave 1 il vincolo sul numero di chiavi minime in un nodo non è rispettato, necessario ridistribuire le chiavi.



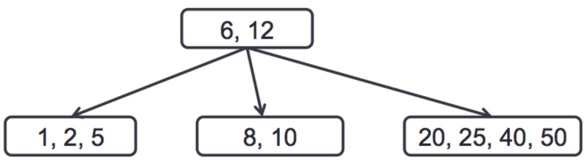
(d) CANC(15): Cancellazione di una chiave non-foglia. Si elimina la chiave e si sposta la chiave 12.



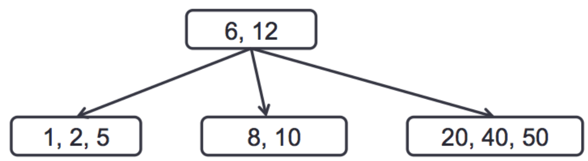
(e) CANC(30): Cancellazione di un nodo non-foglia. Si elimina la chiave e si sposta la chiave 25. Necessario poi ridistribuire le chiavi per rispettare il vincolo sul numero di chiavi.



(f) INS(1): Chiave 1 aggiunta nel rispetto dei vincoli Btree. Non necessarie altre operazioni.



(g) CANC(60): Cancellando la chiave 60 il vincolo sul numero di chiavi minime in un nodo non è rispettato, necessaria un'operazione di fusione.



(h) CANC(25): Chiave 25 cancellata nel rispetto dei vincoli Btree. Non necessarie altre operazioni.

3. Progettare un algoritmo che dato un vettore di interi $A[1..n]$ ed un intero K indica se A contiene valori che sommati danno K, ovvero esistono $A[i_1], \ldots, A[i_m]$ tali che $\sum_{w=1}^{m} A[i_w] = K$.

Soluzione Il problema può essere risolto tramite programmazione dinamica risolvendo i sottoproblemi $P(i, j)$, con $i \in [1..n]$ e $i \in [1..K]$, in cui si considerano solo i primi i valori nel vettore A, ed il valore j invece di k. Il problema originario coincide quindi con $P(n, K)$. I problemi possono essere risolti come segue:

$$P(i,j) = \begin{cases} \textit{true} & \text{se } i = 1 \text{ e } A[1] = j \\ \textit{false} & \text{se } i = 1 \text{ e } A[1] \neq j \\ \text{P(i-1,j)} & \text{se } i > 1 \text{ e } j < A[i] \\ \text{P(i-1,j) or P(i-1,j-A[i])} & \text{se } i > 1 \text{ e } j \geq A[i] \end{cases}$$

Infatti, per raggiungere la somma j usando i primi i numeri del vettore, ci sono essenzialmente due modi: non si utilizza l'i-esimo numero e quindi si raggiunge la somma j usando i primi $i - 1$ numeri, oppure si utilizza l'i-esimo numero e si raggiunge la somma $j - A[i]$ usando i primi $i - 1$ numeri. Il seguente algoritmo risolve tutti i sottoproblemi incrementando l'indice i, ed usa una matrice $M[1..n, 1..K]$ di booleani per memorizzare tali soluzioni. Al termine restituisce $M[n, K]$.

```
algoritmo sommatoria(A[1..n]: Int, K: Int) --> Boolean
    Boolean M[1..n, 1..K]
    // inizializzazione prima riga di M
    for j = 1..K
        if (j == A[1])
```

```
            then M[1,j] = true
            else M[1,j] = false
        endif
    endfor
    // riempimento delle successive righe di M
    for i = 2..n
        for j = 1..K
            if (j < A[i]) then M[i,j] = M[i-1,j]
            else M[i,j] = M[i-1,j] || M[i-1,j-A[i]]
            endif
        endfor
    endfor

    // restituisce la soluzione all'ultimo dei sottoproblemi
    return M[n,K]
```

L'algoritmo esegue un numero costante di operazioni atomiche (di costo costante) per ogni cella della matrice; ha quindi complessità $O(n \times K)$.

4. Si consideri un impianto di irrigazione che collega delle piante a vari rubinetti che possono erogare acqua. Tutti i rubinetti sono inizialmente chiusi, e bisogna capire quale rubinetto aprire per far arrivare più velocemente possibile l'acqua ad una data pianta che necessita di essere annaffiata. L'impianto è rappresentato tramite un grafo non orientato pesato $G = (V, E, w)$ in cui i vertici in V rappresentano rubinetti o piante, un arco $(u, v) \in E$ rappresenta un tubo di collegamento dal vertice u al vertice v, ed il peso $w(u, v)$ indica il tempo che l'acqua impiega per attraversare il tubo (u, v) (sotto l'assunzione che l'acqua impiega il medesimo tempo ad attraversare il tubo partendo dal vertice u o partendo dal vertice v). Progettare un algoritmo che dato il grafo non orientato pesato $G = (V, E, w)$, l'insieme $R \subseteq V$ dei rubinetti, e la pianta $p \in V$ da annaffiare, restituisce il rubinetto $r \in R$ da aprire per far arrivare il più velocemente possibile l'acqua alla pianta p.

Soluzione Il problema prevede di trovare il vertice appartenente ad R che ha il cammino minore per raggiungere il vertice p. Essendo il grafo non orientato, questo coincide con il vertice appartenente ad R a distanza minima da p. I pesi saranno non negativi in quanto quantificano degli intervalli di tempo, quindi è possibile utilizzare l'algoritmo di Dikstra. Visto che tale algoritmo visita i nodi in ordine di distanza da p, è possibile interrompere l'esecuzione appena si raggiunge un nodo appartenente ad R. Se si termina l'algoritmo senza raggiungere nodi appartenenti ad R, allora non è possibile annaffiare tale pianta e si restituisce un errore.

```
algoritmo annaffia(Graph(V,E,w): G, Set[Edge]:R, Edge: p) --> Edge
    // inizializzazione strutture dati
    int n = G.numNodi()
    double D[1..n]
    for v = 1..n do
        D[v] = INFINITY
    endfor
    D[p] = 0
    CodaPriorita<int, double> Q; Q.insert(p, D[s]);

    // esecuzione algoritmi di Dijkstra
    while (not Q.isEmpty()) do
        u = Q.find(); Q.deleteMin()
        if (u in R) return u
        for each v adiacente a u do
            if (D[v] == INFINITY) then
                // prima volta che si incontra v
                D[v] = D[u] + w(u,v)
                Q.insert(v, D[v]);
            elseif (D[u] + w(u,v) < D[v]) then
```

```
                // scoperta di un cammino migliore per raggiungere v
                Q.decreaseKey(v, D[v] - D[u] - w(u,v))
                D[v] = D[u] + w(u,v)
            endif
        endfor
    endwhile
    return error
```

La complessità nel caso pessimo coincide con la complessità dell'algoritmo di Dijkstra, ovvero $O(m \times \log n)$ dove $m = |\mathtt{E}|$ e $n = |\mathtt{V}|$.

The End.